2017

Genetic optimisations for satisfiability and Ramsey theory

Barnes, A.

Barnes, A. (2017) 'Genetic optimisations for satisfiability and Ramsey theory', The Plymouth Student Scientist, 10(2), p. 193-207.
http://hdl.handle.net/10026.1/14165

The Plymouth Student Scientist
University of Plymouth

*The Plymouth Student Scientist, 2017, **10** (2), 193-207*

Genetic optimisations for satisfiability and Ramsey theory

Andrew Barnes

Project Advisor: Angelo Cangelosi, School of Computing, Electronics and Mathematics, Plymouth University, Drake Circus, Plymouth, PL4 8AA

Abstract

The art of using evolutionary mechanisms for identifying satisfiability has produced a range of efficient solutions to this otherwise computationally challenging problem. Since their first use these evolutionary methods have been changed and adapted to produce increasingly efficient solutions. This paper introduces two unique alternatives to the optimisation of these methods, the first through the introduction of alternative mutation operators and the second through utilizing a grammatical encoding which has been proven to improve neuroevolution. The goal of this paper is to identify whether these two alternatives are candidates for future investigation in improving evolutionary satisfiability solvers.

Introduction

The satisfiability problem

The Boolean satisfiability problem (SAT) is a problem in Boolean algebra which is to prove whether a given expression (in conjunctive normal form) can be satisfied. A simple example of a satisfiable equation is $(\bar{A} + A)$; we can see this must be satisfiable due to its equivalent truth table (

A	$\bar{A}$	$Result$
1	0	1
0	1	1

Figure 1); however, if we take an unsatisfiable equation $(\bar{A}.A)$ we can prove this to be unsatisfiable in the same way (

Figure **2**).

Figure 1: Truth Table for a Satisfiable Equation.

A	$\bar{A}$	$Result$
1	0	0
0	1	0

Figure 2: Truth Table for an Unsatisfiable Equation

This problem is exceptionally important in the domain of computational complexity as it is both the first and the simplest problem which has been proven to be NP-complete [1]. Being NP-complete means that the problem both exists in the complexity class of NP and can be reduced to by all other problems in NP under certain reduction rules as presented by Aaronson [2]; this also implies that unless $P = NP$ [1] the worst-case time-complexity for SAT algorithms will be $O(n^k)$. The SAT problem however is not only interesting in theoretical computer science but its applications are broad; for example, Boyarski, Stern & Surynek [3] used SAT solving to enhance pathfinding in multiagent systems.

Ramsey numbers

As best presented by example, Ramsey's numbers give the solution to a popular combinatorics problem in mathematics; the problem being: A party is being planned; how many people must be invited to guarantee that at least m people will know each other or n will not know each other? [4]. In graph theory, a Ramsey number,

$R = (m, n)$, is the minimum number of vertices (v) to guarantee that all graphs of order v must have a clique of order m and an independent set of order n. For example, one such Ramsey number $R = (3, 3)$ we know to be 6 as proven by Greenwood and Gleason [5]; as shown in Figure 3 we cannot guarantee a colouring on a graph of order 3, 4 or 5. However, on a graph of order 6 we cannot escape guaranteeing we have at least one clique of size m or n; so to answer our original problem to throw a party with at least three people either knowing each other or three people not knowing each other there must be at least six people.

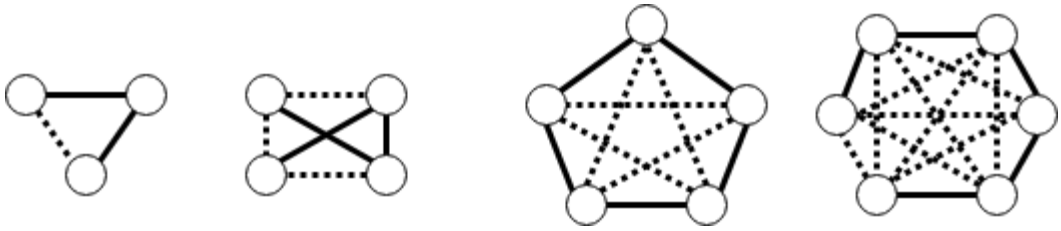

Figure 3: Ramsey Number Visualisation.

Genetic computation

Genetic computation can be traced back to Turing's work in 1948 where he first proposed the idea of an evolutionary or genetics based search method [6]; the earlier biological work of Darwin inspired this idea [7]. The core idea is that there exists a population of organisms in which only the fittest individuals would survive and continue to live on through generations whereas weaker organisms would be removed. This phenomenon is what is today known as survival of the fittest. Work in artificial evolution however did not start until the 1950s [8] when there were several groups investigating its applications. For example, Rechenberg [9] utilized evolution to optimize real-value pairs for aerospace devices; whereas, Holland [10] was working on a method he termed as "Genetic Algorithms" (GA).

Within each algorithm is a population of potential solutions to a given problem; these solutions generally being represented as a string of bits. In biological terms, this string of bits is referred to as a genome with each bit being called a gene. This population then undergoes three genetic operators which can take a variety of forms as can be found in various literatures [6] [8] [11]. However, the goal of each of these operators is the same: Selection, Reproduction, and Mutation; selection selects the best genomes in the population for reproduction (sometimes called crossover) before finally the resulting population is mutated to produce a new population. This process is then repeated until either it is stopped or a suitable solution is found. An example of this process is shown in Figure 4.

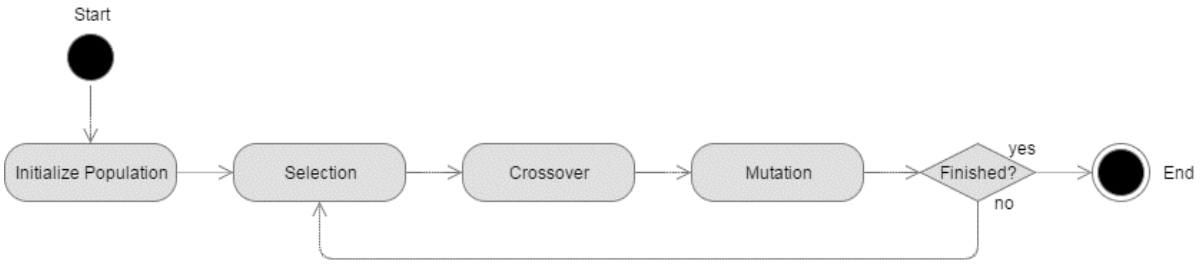

Figure 4: A Genetic Algorithm.

Project Overview

This project looks to combine the three topics discussed above to optimize genetic algorithms to find solutions to Boolean equations through a modified party problem. The problem this project investigates is as follows, given a complete graph of size n and a clique size of x does there exist a graph which doesn't contain a clique of x vertices which are either connected or disconnected from each other. To do this the project will investigate the conversion of this problem to a Boolean equation and then investigate the effects of altering the mutation methods of two key SAT genetic algorithms (EvoSAP [12]and FlipGA [13]); it then looks to compare two blind genetic algorithms using different encoding methods (direct encoding and morphogenetic encoding). This report begins by introducing the current literature on genetic algorithms for satisfiability and the various genetic encodings currently in use. It then describes how the primary project objectives identified in **Error! Reference source not found.**, before continuing to describe how these are met through development of a simulator which shall be used to carry out the required experiments. Finally, this report concludes on the results of the experiments.

Methods for finding satisfiability

Traditional methods

Traditional techniques for SAT rely heavily on local-search based approaches. For example, the most widely used procedure, which is known as the Davis-Putnam procedure [14] utilizes a search over the given formula for unsatisfiable clauses. This procedure has acted as the backbone for many other variants such as that presented by Selman, Mitchell and Levesque [15] which disregards part of the search procedure in favour for a recursive search method instead.

Older work on an algorithm known as GSAT [16] utilizes a greedy-local search heuristic, the method looks to iteratively improve an initial solution which is like genetic methods function but with less random variables. GSAT has been proven to outperform the David-Putney by at least one order of magnitude. However, the author admits the GSAT method is incomplete and there is no guarantee of finding a solution but mentions that artificial intelligence methods are one such technique which could be promising in terms of time complexity. However, recent successes in proving satisfiability have continued to use a purely local search based approach such as the cube and conquer method presented by Kullman *et al.* [17] which has been used to verify and solve Pythagorean triples [18].

Evolutionary approaches

Evolutionary and genetic approaches to satisfiability solving emerged following the Davis-Putnam procedure; these methods ranged from the purely evolutionary methods such as the Stepwise Adaptation of Weights method (SAWEA) [19] to hybrid methods incorporating a lightweight local search procedure such as FlipGA [13] and EvoSAP [12]. However, the effect this has on results is big; as presented by Gottlieb and Voss [20] the results showed that FlipGA had success rate (SR) which was 11%

greater than SAWEA, but it also did so in 70.8% fewer average evaluations per solution (AES).

The biggest challenge with evolutionary approaches to SAT is how to calculate a fitness value which represents how close a given organism is to a satisfiable solution. The most popular of these fitness functions is MAXSAT as used in [13] [12]; here the algorithms look at each clause in the equation and identify whether that clause is satisfied or not, we can then calculate the total fitness for a given organism based on the number of clauses satisfied divided by the total number of clauses. Similarly, SAWEA [19] utilizes an adaptive fitness function based on MAXSAT; in SAWEA each clause is given a weighting which makes the fitness function: where n is the number of clauses. During evolution, the weights are adjusted according to the following function: where y is the fittest organism in the population; this increases the importance (weights) of those clauses which are currently unsatisfied.

Furthermore, the mutation operators employed in the different methods are not used as a variable in many experiments. For example, in SAWEA [19] the algorithm is designed to use a method called MutOne in which a single gene in each organism is mutated randomly whereas in other literature [13] [12] [21] they stick to random mutations. However, even in these random mutations differences are clear; for example, in FlipGA each gene in the population has a chance of being mutated, whereas EvoSAP uses an organism by organism random chance of being mutated before selecting genes.

Due to the differing nature of these fitness functions it is important that measurements are taken when organisms are evaluated and/or flipped. As described by Gottlieb, Marchiori and Rossi [21] the three quantitative measurements which can be used to analyse the complexity of each approach are as follows: the average evaluations per solution (AES), average flips and evaluations per solution (AFES) and average flips per solution (AFS). These measurements as opposed to the actual run-time of the algorithms allow researchers to compare results without the need to worry about implementation detail such as language or hardware. They also go further to mention that although the local search methods run-time would be a measurement of complexity, in the case of genetic and evolutionary methods this is hardly the case as it is machine and implementation dependant.

Genetic encodings

Variable string

Commonly described as the most intuitive ways of encoding these solutions [21], representing the solution to a Boolean equation as a bit string. For example, given a Boolean function b which has n variables a solution bit-string would contain n bits. With this representation, it would be easy to assume a good fitness function for such

an encoding would be the function b; however, as stated by Gottlieb, Marchiori and Rossi [21] this will cause the algorithm to fall back to a random search.

To rectify this issue, a fitness function introduced by De Jong and Spears [22] called $MAXSAT$ takes advantage of equations which are in conjunctive normal form [23] to analyse how many clauses in the equation are satisfied. One example is given an equation $b = (A + B).(C + D)$ and an input string of $i_1 = 1000$ the equation would equate as follows: $b = (1 + 0).(0 + 0) = (1).(0)$ Here we can see we have satisfied a single clause of the equation and so the fitness of i_1 is 0.5 (50%) whereas the input string $i_2 = 1010$ would have a fitness of 1 (100%) as it satisfies both clauses.

Real-Values

Another such way of representing these solutions comes in the form of real-values which are inspired by Montana's work on evolving and training neural networks [24]. The concept is to have each gene consist of a floating-point number n such that $n \in \mathbb{R}$. Although originally presented by Montana in 1989 it wasn't until 1998 that it was applied to satisfiability problems by Back *et al*. [25]. This then converts a satisfiability problem into continuous optimisation problems [21]. To do this they swap each variable x_i and its inverse $\bar{x_i}$ with a floating-point equivalent between -1 and 1 such that $\bar{x_i} = (y_i + 1)^2$ and $x_i = (y_i - 1)^2$. They then replace the logical · by an arithmetic $+$ and the logical $+$ with an arithmetic $\times$. This then produces a continuous function which can be optimised.

$$eq_{boolean} = (A \cdot B \cdot \bar{C}) + (\bar{A})$$

$$eq_{continuous} = ((y_A - 1)^2 + (y_B - 1)^2 + (y_C + 1)^2) \times ((y_A + 1)^2)$$

As you can see above, the Boolean variables are converted and evolved as floating-point numbers represented as y_X, according to their logical sign they are then converted for evaluation. This encoding however does not show much promise to improving SAT solving algorithms; as found by Back *et al*. [25] and Eiben and van der Hauw [19] this floating-point representation is significantly inferior to the previously mentioned SAWEA method.

Pathing

As presented by Gottlieb and Voss [20] another way to take advantage of the conjunctive normal form nature of SAT problems is that a satisfying solution must satisfy at least one variable in each clause of the equation. For example, given $b = (A + B).(C + \bar{A})$ one such path would be (A, C) which is feasible whereas a path of $(A, \bar{A})$ is infeasible as it contains both a variable and its inverse. Moreover, as presented by Gottlieb, Marchiori and Rossi [21] this representation is far more compact than alternatives such as variable string encodings; however, as found by the representation's original authors Gottlieb and Voss the representation does not produce results which are close to that of a variable encoding.

Mutation method optimisation

Hypothesis

How does changing the mutation method affect the performance of hybrid-SAT genetic algorithms? Due to the nature of the FlipGA methods introducing mutation method 1 would increase the rate of mutation (from 0.45 [13] to 0.81 per gene) which when applied along with the local search on a population will considerably increase the search space, decreasing the number of evaluations it will require.

The same comment would apply for EvoSAP; however, in the case of EvoSAP there is no reproduction as such so more emphasis is placed on the local-search finding a solution. This could however lead to a greater success rate as mutation method 1 will reduce the mutation rate of each gene in EvoSAP from 0.9 [12] to 0.81. For the second mutation method, the probability of each gene mutating in FlipGA stays consistent at 0.45 and so a change in the results would appear unlikely. However, for EvoSAP the chance of a gene mutating is halved from 0.9 to 0.45 and so I would expect more extreme results than those found for EvoSAP with the first mutation method.

Design

To ensure consistency in results the candidate mutation methods will be used on two separate evolutionary procedures for finding satisfiable solutions. Further to this a benchmark for each of the two methods is taken beforehand for comparison with the results. The key variable is the mutation method which acts as a core component of the two hybrid SAT genetic algorithms. The mutation method shall be varied between the original mutation methods of both algorithms as well as the two methods presented below in Figure ; as presented in their original papers each algorithm will be run with an ideal mutation rate of 0.9.

```
MUTATION METHOD 1
BEGIN
 i = 0
 WHILE i < length(population)
  BEGIN
   ro = Random (0<= x <= 1)
   IF ro < MUTATION_RATE
    THEN
     v = 0
     WHILE v < length(variables)
      BEGIN
       rv = Random (0<= x <= 1)
       IF rv < MUTATION_RATE
        THEN
         Flip v in population[i]
        END
       v = v + 1
      END
    END
   i = i + 1
  END
END
```

```
MUTATION METHOD 2
BEGIN
 no_vars = length(variables)
 i = 0
 WHILE i < length(population)
  BEGIN
   ro = Random (0<= x <= 1)
   IF ro < MUTATION_RATE
    THEN
     rs = random (0<= x <= no_vars - 1)
     j = 0
     WHILE cnt <= r
      BEGIN
       Flip position j in population[i]
       j = j + 1
      END
    END
   i = i + 1
  END
END
```

Figure 5: Mutation Methods. This figure illustrates the two variable mutation methods to be evaluated as part of the Mutation Method Optimisation experiment.

Method

A simulator was developed as part of this experiment. To ensure reliable results all code is available with full unit tests [26]. Once proven reliable the simulator was moved onto a secure server at the University of Plymouth (Eeyore) and checked for a second time to ensure all tests were passing locally. The next step was to mitigate any risks to this experiment and as presented in Table 1 there was the potential to mitigate the risks and the damage caused.

Table 1: Mutation Method Optimisation Risks.

ID	Name	Mitigation Measures
1	Server Faults (Powercut, restarts, turn-offs)	To mitigate the loss of results after each mutation method has been tested its results will be saved to disk.
2	Code Failure (Runtime errors)	Thorough unit-tests have been implemented and must be passing before experiments are run.
3	Criminal Activity (Tampering of results, stopping the experiments.	The experiments shall be run within a Windows environment protected by a password.
4	Algorithms Stopping prematurely	The number of generations the algorithms shall run to at max is the same as presented in similar literatures 1,000 [13].

Running the test will utilize part of the DIMACS benchmark SAT set [27]; namely the first 50 instances available in the *CBS_k3_n100_m441_b70* data set. Each instance consists of 100 Boolean variables and 441 clauses. Each instance shall be trialled twenty times to produce a reliable result and the results shall be averaged out.

Results

Presented in Table 2 is an overview of the results for the different mutation methods with all three mutation methods and below this in Figure , Figure and Figure is a graphical representation of the results.

Table 2: Results of Mutation Experiment.

Average Evaluations Solution (AES)			Success Rate		
	FlipGA	**EvoSAP**		**FlipGA**	**EvoSAP**
Mutation 1	127588.78	67700.02	**Mutation 1**	0.986	0.692
Mutation 2	70251.26	37355.562	**Mutation 2**	0.995	0.882
Original	44326.31373	25835.56667	**Original**	0.9980392157	0.4509803922

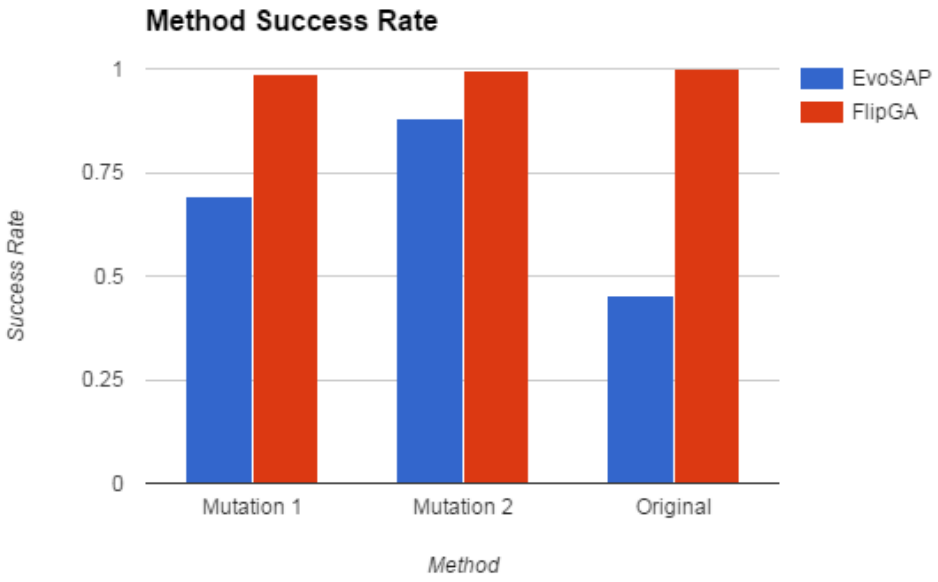

Figure 6: Mutation Method Success Rate. This figure illustrates the success rate of each mutation method for both algorithms across the data set.

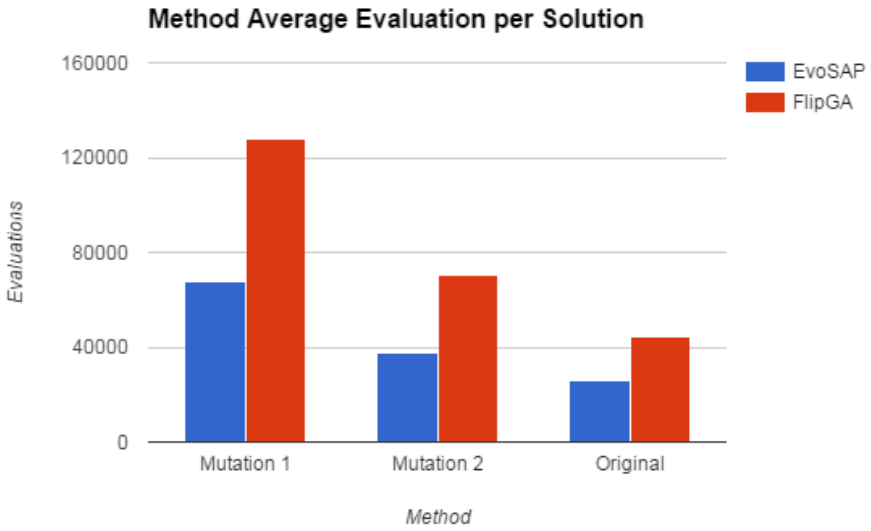

Figure 7: Mutation Method AES. This figure illustrates the AES of each mutation method for both algorithms across the data set.

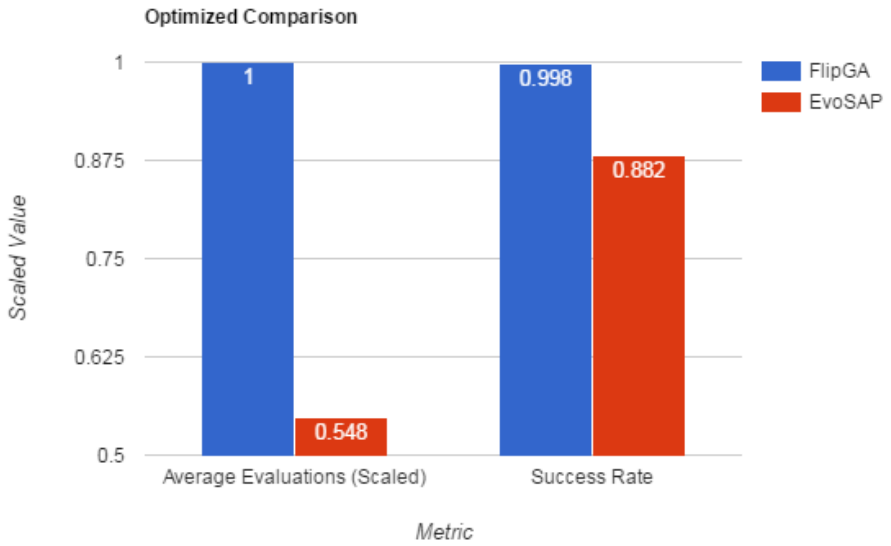

Figure 8: Optimised Mutation Comparison. This figure illustrates the difference between a scaled number of evaluations to the success rate for both optimised algorithms of EvoSAP and FlipGA.

Conclusion

Contrary to the original predictions, FlipGA performed worse with the higher mutation rates provided by both method 1 and method 2, both of which lowered the success rate and increased the number of evaluations required. However, despite mutation method 2 providing the same probability of a gene mutating as its original method the method still performed marginally worse.

Despite this, the inverse appears true for EvoSAP; as you can see in Figure 6 both mutation methods increased the success rate although at a cost of increasing the number of evaluations required. Nonetheless, mutation method 2 improved EvoSAP considerably at minimal cost to the number of evaluations. One clear conclusion which can be drawn from these results is that in utilizing a mutation method with a lower chance of mutating each gene improves these algorithms significantly; putting more emphasis on the local search and in FlipGA's case the reproduction elements.

Moreover, on comparing the differences between the best mutation methods for EvoSAP and FlipGA there is a significant difference in the number of evaluations required compared to the success rate of each method. As shown in Figure FlipGA close to doubles the number of evaluations required for a 0.116 increased chance of finding a solution. What these results show is a clear trade-off in the number of evaluations allowed and the required success rate; EvoSAP with this second mutation method provides a better proportion of success rate to number of evaluations.

Morphogenetic encoding experiment

Hypothesis

Does a Morphogenetic encoding scheme increase the efficiency of a purely genetic algorithm for clique problems through SAT? Due to the nature of the morphogenetic implementation the mutations mutate a larger amount of the chromosome in comparison with a single mutation in a direct encoding. This following the results of the previous experiment led to a prediction that through widening the random search space the GA utilizing morphogenetic encoding will reach the global maximum quicker than the equivalent GA using direct encoding.

Design

In this experiment, a simple blind genetic algorithm is used as the control method with a benchmark taken before modifications to the encoding takes place. The only variable in this experiment is the encoding type used; the design of the genetic algorithm is simple and consists of a population of size 10 and a mutation rate of 0.9. The genetic operators are described in Table3 and the two encoding types are described in Table 4.

Table 3: BlindGA Genetic Operators.

Operator	Description
Selection	Selection is done using the *MAXSAT* function [22] and produces two parents which can be used for this generation.
Reproduction	Single point crossover is achieved using a random crossover point between the two parents.
Mutation	Each organism is given a mutated at a rate of 0.9 and each gene inside a mutated organism is modified at a rate of 0.5.

Table 4: Overview of Encodings.

Encoding	Description
Direct	Utilizing the variable string encoding as done by various other literatures [19] [22].
Morphogenetic	Adapting the graph generation system proposed by Kitano [28].

Method

Similarly, to the experiment discussed above the code for this experiment was also verified using the unit-tests included in the simulator and were run on the same university server. The risks for this experiment are presented in Table below; a key risk to note is no.5 which was identified during the running of the first experiment, automatic security updates stopped all processes being used by the simulator.

Table 5: Encoding Evaluation Risks.

ID	Name	Mitigation Measures
1	Server Faults (Powercut, restarts, turn-offs)	To mitigate the loss of results after each mutation method has been tested its results will be saved to disk.

2	Code Failure (Runtime errors)	Thorough unit-tests have been implemented and must be passing before experiments are run.
3	Criminal Activity (Tampering of results, stopping the experiments.	The experiments shall be run within a Windows environment protected by a password.
4	Algorithms Stopping prematurely	The number of generations the algorithms shall run to at max is the same as presented in similar literatures 1,000 [13].
5	Automatic Security Updates	Agreement was made to turn-off automatic security updates during the running of this experiment.

Running the test will utilize instead of a SAT data set a selection of clique problems converted to SAT problems. The definitions of each of these problems can be found in Table ; the problems are to be read as given a graph of size *graph size* is there such a colouring of the edges that no clique of size *clique size* exists. For each of the problems the algorithms will have 20 trials and their results averaged to give a score.

Table 6: Encoding Experiment Data.

Graph Size	Clique Size
3, 4, 5	3
4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17	4

Results

Presented in Table is an overview of the results for the two encoding methods; below this you will find Figure and Figure which illustrate this information.

Table 7: Encoding Results Overview.

Total AES		Success Rate	
Direct Encoding	**Kitano Encoding**	**Direct Encoding**	**Kitano Encoding**
14366.55	2992	0.6117647059	0.6470588235

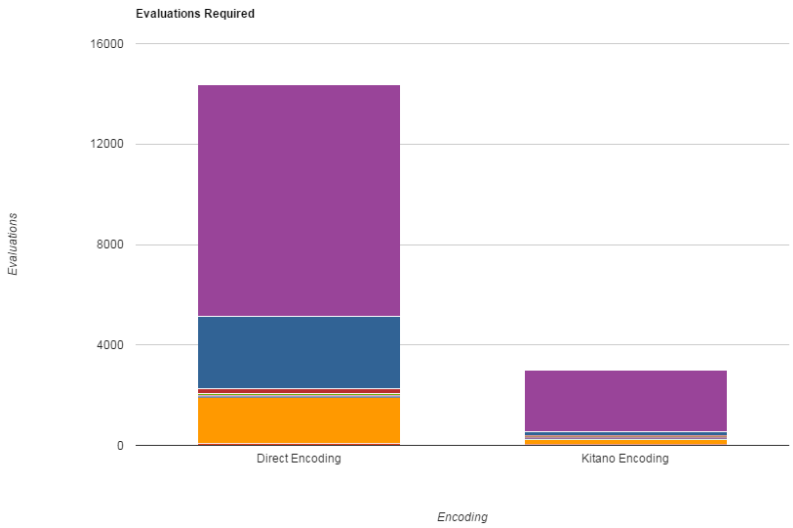

Figure 9: Encoding Evaluations Results. This figure illustrates the differences between the number of evaluations required by both the direct and morphogenetic encodings.

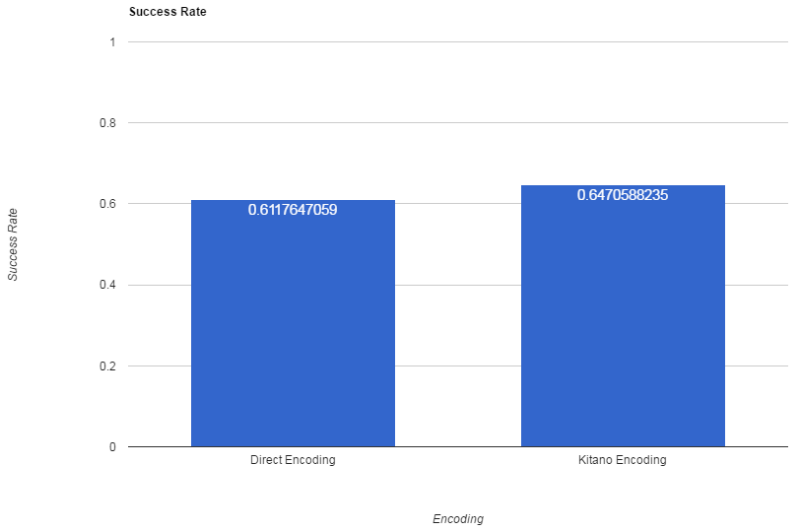

Figure 10: Encoding Success Rate Results. This figure illustrates the differences between the success rates of the two encodings.

Conclusion

Matching the original predictions made, the morphogenetic encoding reduced the number of evaluations required by the genetic algorithm by 80% while also marginally increasing the success rate. What is also apparent from the experiment is that the morphogenetic approach may also increase the reliability of the result whereas the direct encoding was shown to be more inconsistent with test case 11.

References

[1] S. Cook, “The Complexity of Theorem-Proving Procedures,” in *In Proceedings of Third Annual ACM Symposium on Theory of Computing*, New York, 1971.

[2] S. Aaronson, “Complexity Zoo,” University of Waterloo, [Online]. Available: https://complexityzoo.uwaterloo.ca/Complexity_Zoo. [Accessed 01 04 2017].

[3] E. Boyarski, R. Stern and P. Surnek, “Boolean Satisfiability Approach to Optimal Multi-Agent Path Finding under the Sum of Costs Objective,” in *In Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, Singapore, 2016.

[4] E. Weisstein, “Ramsey Number,” Wolfram, [Online]. Available: http://mathworld.wolfram.com/RamseyNumber.html. [Accessed 01 04 2017].

[5] R. E. Greenwood, “Combinatorial Relations and Chromatic Graphs,” *Canadian Journal of Mathematics,* vol. 7, pp. 1-7, 1955.

[6] A. E. Eiben and J. E. Smith, Introduction to Evolutionary Computing, Berlin: Springer-Verlag Berlin and Heidelberg, 2010.

[7] C. Darwin, The Origin of Species, New York: D.Appleton and Company, 1923.

[8] M. Mitchell, An Introduction to Genetic Algorithms, Cambridge, MA: The MIT Press, 1999.

[9] I. Rechenberg, “Evolution Strategy: Nature's Way of Optimization,” *Lecture Notes in Engineering,* vol. 47, pp. 106-126, 1965.

[10] J. H. Holland, Adaptation in natural and artificial systems, Cambridge, MA: The MIT Press, 1975.

[11] S. Kwong, K. F. Man and K. S. Tang, Genetic Algorithms, London: Springer, 1999.

[12] E. Marchiori, C. Rossi and J. N. Kok, “An adaptive evolutionary algorithm for the satisfiability problem,” in *Proceedings of the 2000 ACM symposium on Applied computing*, New York, 2000.

[13] E. Marchiori and C. Rossi, “A Flipping Genetic Algorithm for Hard 3-SAT Problems,” in *Proceeding GECCO'99 Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation*, San Francisco, 1999.

[14] M. Davis, G. Logemann and D. Loveland, “A Machine Program for Theorem Proving,” *Communications of the ACM,* pp. 394-397, 07 1962.

[15] B. Selman, D. Mitchell and H. Levesque, “Generating Hard Satisfiability Problems,” *Artificial Intelligence,* vol. 81, no. 1, pp. 17-29, 1996.

[16] H. Levesque, D. Mitchell and B. Selman, “A New Method for Solving Hard Satisfiability Problems,” in *Proceedings of the Tenth National Conference on Artificial Intelligence*, Menlo Park, 1992.

[17] A. Biere, M. Heule, O. Kullman and S. Wieringa, “Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads,” in *Proceedings of the 7th International Haifi*

Verification Conference on Hardware and Software: Verification and Testing, Berlin, 2011.

[18] M. Heule, O. Kullman and V. Marek, “Solving and Verifying the Boolean Pythagorean Triples Problem via Cub-and-Conquer,” in *19th International Conference on Theory and Applications of Satisfiability Testing*, Bordeaux, 2016.

[19] A. Eiben and J. Van Der Hauw, “Solving 3-SAT with adaptive genetic algorithms,” in *Proceedings of the Fourth IEEE Conference on Evolutionary Computation*, Piscataway, 1997.

[20] J. Gottlieb and N. Voss, “Representations, Fitness Functions and Genetic Operators for the Satisfiability Problem,” in *Proceedings of the third european conference on artificial evolution*, London, 1997.

[21] J. Gottlieb, E. Marchiori and C. Rossi, “Evolutionary Algorithms for the Satisfiability Problem,” *Evolutionary Computing,* vol. 1, no. 1, 2002.

[22] K. A. De Jong and W. M. Spears, “Using Genetic Algorithms to solve NP-Complete Problems,” in *Proceedings of the third international conference on Genetic Algorithms*, San Francisoco, 1989.

[23] E. W. Weisstein, “Conjunctive Normal Form,” Wolfram, [Online]. Available: http://mathworld.wolfram.com/ConjunctiveNormalForm.html. [Accessed 20 04 2017].

[24] D. Montana and L. Davis, “Training Feedforward Neural Networks Using Genetic Algorithms,” in *Proceedings of the 11th International Join Conference on Artificial Intelligence*, Detroit, 1989.

[25] T. Back, A. Eiben and M. Vink, “A Superior Evolutionary Algorithm for 3-SAT,” in *Proceedings of the 7th International Conference on Evolutionary Programming*, London, 1998.

[26] A. P. Barnes, “Evolutionary Party Problem Simulator,” GitHub, [Online]. Available: https://github.com/Sciprios/EvolutionaryPartyProblemSimulator. [Accessed 01 04 2017].

[27] University of British Columbia, “SATLIB - Benchmark Problems,” [Online]. Available: http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html. [Accessed 2017 04 23].

[28] H. Kitano, “Designing neural networks using genetic algorithm with graph generation system,” *Complex Systems,* vol. 4, pp. 461-476, 1990.